Sensor network for environmental data based on Bluetooth LE sensors

Sensor network for environmental data based on Bluetooth LE sensors

So in 2018 it was time to build a sensor network system:

- Manages the sensors, collects the data from the sensors
- Is widely configurable, sensors can be added, removed and reconfigured by external configuration
- Provide data aggregation, calibration
- Offers flexible storage options
- Provide switching of external electrical consumers
- Based on IoT
- Running on a mini computer like Raspberry Pi.
- All software development based on .net framework

Requirements for the sensors

- Size of sensors, should be small
- Power supply, should be powered by battery or any other wireless power supply
- Price
- Easily configurable, so that a sensor can be added and removed to the system
- Standardized communication
- No additional hardware or software necessary

First look on “IP” based solutions and sensors

But most of them:

- Too big in size
- Too expensive
- The communication was based on undocumented proprietary protocols
- For hacking a sensor special hardware like ZigBee sniffers was required
- “Mothership” or access to a vendor specific cloud is required to sniff the communication

Bluetooth LE Sensors

- They are small
- The protocols are standardized
- They are easy to configure
- Low power consumption, can be powered by batteries
- They are cheap (temperature sensors start at around 10 $)
- They are available for many types of environmental data
- In most cases easy to hack

Bluetooth Low Energy (Bluetooth LE, Colloquially BLE)

- Bluetooth Low Energy is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group (Bluetooth SIG) aimed at novel applications in the healthcare, fitness, beacons, security, and home entertainment industries. Compared to Classic Bluetooth, Bluetooth Low Energy is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range.
  In contrast to Classic Bluetooth, Bluetooth Low Energy (BLE) is designed to provide significantly lower power consumption. This allows Android apps to communicate with BLE devices that have stricter power requirements, such as proximity sensors, heart rate monitors, and fitness devices.

Types of Bluetooth LE sensors

- Gatt Generic Attributes (GATT) sensors with paired and unpaired mode
- Advertising Beacons

Bluetooth LE Basics

- Every Bluetooth LE device has a unique MAC address, also called Bluetooth address
- The Range of the Bluetooth LE signal can be up to 450 m
- All Bluetooth communication starts with an advertising packet or frame of max. 31 bytes send from the BLE device
- The client then can establish a Gatt communication in paired or unpaired mode
- Or use the data of the advertising frame

GATT Specifications

- Generic Attributes (GATT) services are collections of characteristics and relationships to other services that encapsulate the behavior of part of a device.
- A GATT profile describes a use case, roles, and general behaviors based on the GATT functionality, enabling extensive innovation while maintaining full interoperability with other Bluetooth® devices.

GATT Specifications

- Generic Attribute Profile (GATT)
  The GATT profile is a general specification for sending and receiving short pieces of data known as "attributes" over a BLE link. The Bluetooth SIG defines many standard profiles for Low Energy devices. A device can implement more than one profile.

- Service
  A service is a collection of characteristics. For example, you could have a service called "Heart Rate Monitor" that includes characteristics such as "heart rate measurement." You can find a list of existing GATT-based profiles and services on bluetooth.org

- Characteristic
  A characteristic contains a single value and 0-n descriptors that describe the characteristic's value. A characteristic can be thought of as a type, analogous to a class.

- GATT services and characteristics are distinguished by their UUID

- GATT Sensors can have multiple services, that can be standard or custom services

Gatt standard profile: Battery level

Uuid service	0000180F-0000-1000-8000-00805F9B34FB
Uuid DataCharacteristic	00002A19-0000-1000-8000-00805F9B34FB

Custom Gatt profile: BLEHome SensorBug 109D7E

Gatt data Luminosity

Uuid service	9DC84838-7619-4F09-A1CE-DDCF63225B20
Uuid config characteristic	9DC84838-7619-4F09-A1CE-DDCF63225B21
Uuid data characteristic	9DC84838-7619-4F09-A1CE-DDCF63225B22
Uuid alert characteristic	9DC84838-7619-4F09-A1CE-DDCF63225B23
Uuid stats characteristic	9DC84838-7619-4F09-A1CE-DDCF63225B24

Before start coding

Target Hardware

- Raspberry Pi 3B, Dragonboard 410C

Operating systems

- Windows IoT Core
- Windows 10

Development Environment

- C#, .net
- UWP
- Visual Studio 2017

Gatt Sensors pros and cons

- Many types of sensors available
- Always 1: 1 Sensor⇔ Consumer
- You can be sure that you receive data from the right sensor
- Data are always public to anybody that can intercept the Bluetooth signal
- Hacks are possible by exploring the services and characteristics of a Gatt sensor

Beyond Gatt beacons

What can be done with Bluetooth LE device beyond Gatt, beyond the complex operations of Gatt?

We have the advertising frames, with a maximum of 31 bytes.

- So, add a payload to advertising frame
- Let the receiving system determine the signal strength of the received frame
- Transmit the advertising frames permanently

Advertising Beacons

Apple

- iBeacon – proximity beacon

Google

- Eddystone Uid – proximity beacon
- Eddystone Url – proximity beacon
- Eddystone Tlm – telemetry beacon

Eddystone TLM frame specification

Frame data type: 0x16

Offset	Byte value
0	0xAA
1	0xFE
2	Payload battery and temperature

Payload starts at offset 2

Offset	Field	Description
0	Frame Type	0x20 = TLM frame type (Url frame type would be 0x10)
1	Version	TLM version, value = 0x00
2	VBATT[0]	Battery voltage, 1 mV/bit
3	VBATT[1]	
4	TEMP[0]	Beacon temperature divided by 256
5	TEMP[1]	
6	ADV_CNT[0]	Advertising PDU count
7	ADV_CNT[1]	
8	ADV_CNT[2]	
9	ADV_CNT[3]	
10	SEC_CNT[0]	Time since power-on or reboot
11	SEC_CNT[1]	
…	…	

Blukii beacon “Swiss knife”

- Temperature
- Humidity
- Luminosity
- Atmospheric pressure
- Acceleration
- Magnetic field
- Battery level

Blukii frame specification

Frame data type: 0xFF GAP_ADTYPE_MANUFACTUR_SPECIFIC

Frame data

Field	Offset	Bytes	Value
Manufacturer ID	0	2	0x024F f
Product ID	2	1	0x02
MAC Adress	3	6	
Hardware Version	9	1	
Firmware Version	10	2	
Battery Level	12	1	0x00 0% – 0x64 100% in 1% steps
Advertising Interval	13	1	0x01 100ms – 0x64 10s
Payload environmental data	14	8	

Payload environmental data starts at offset 15

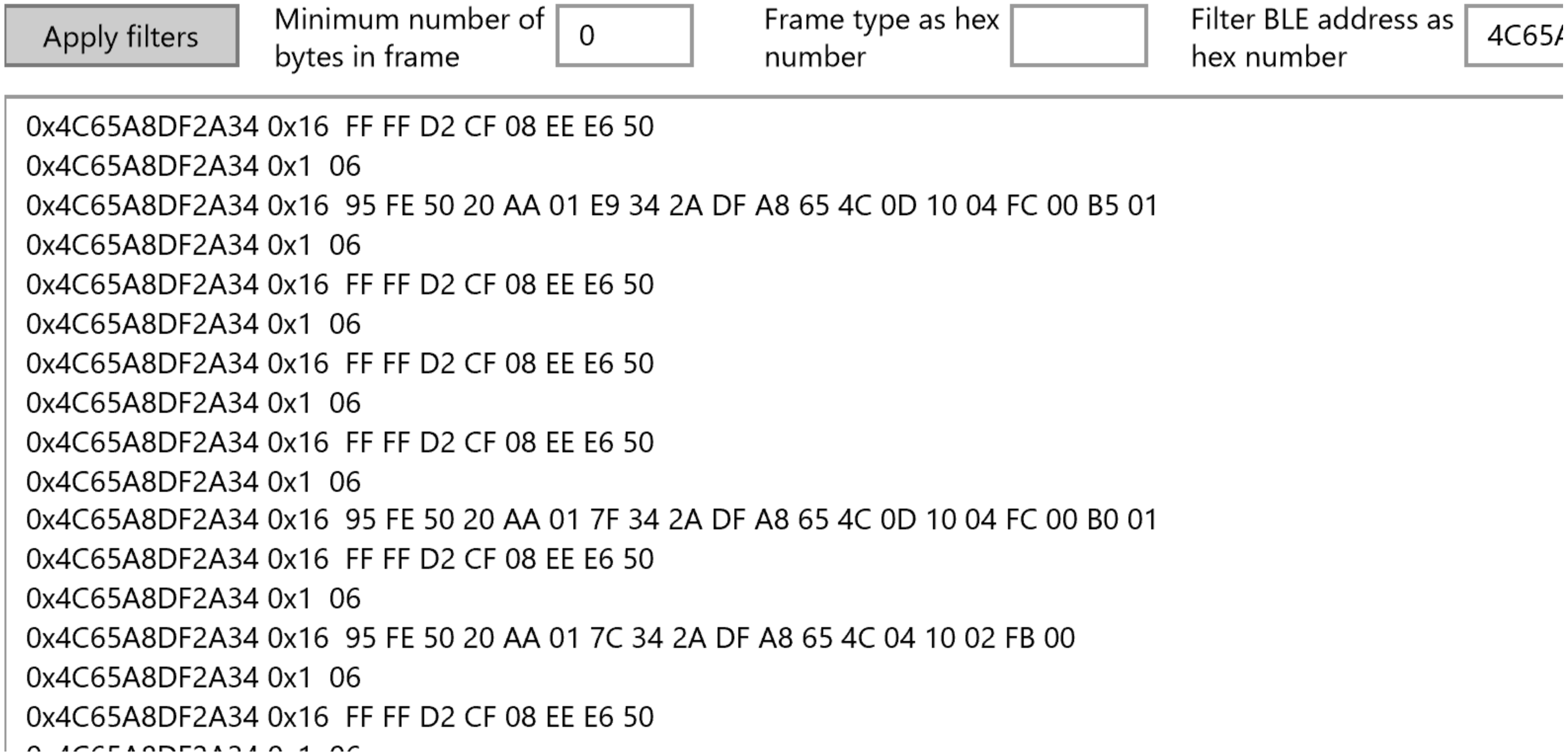
Offset	Bytes	Description
0	1	0x10 = Frame Type Environment
1	2	Atmospheric pressure in dPa as 16 unsigned integer
3	2	Luminosity (lx) as 16 Bit unsigned integer.
5	1	Humidity in % as 8 bit unsigned integer
6	2	Temperature in °C as decimal number byte 6 pre decimal byte 7 decimal

Explore a unknown device without a spec

Original Xiaomi Mijia Bluetooth Temperature Humidity Sensor

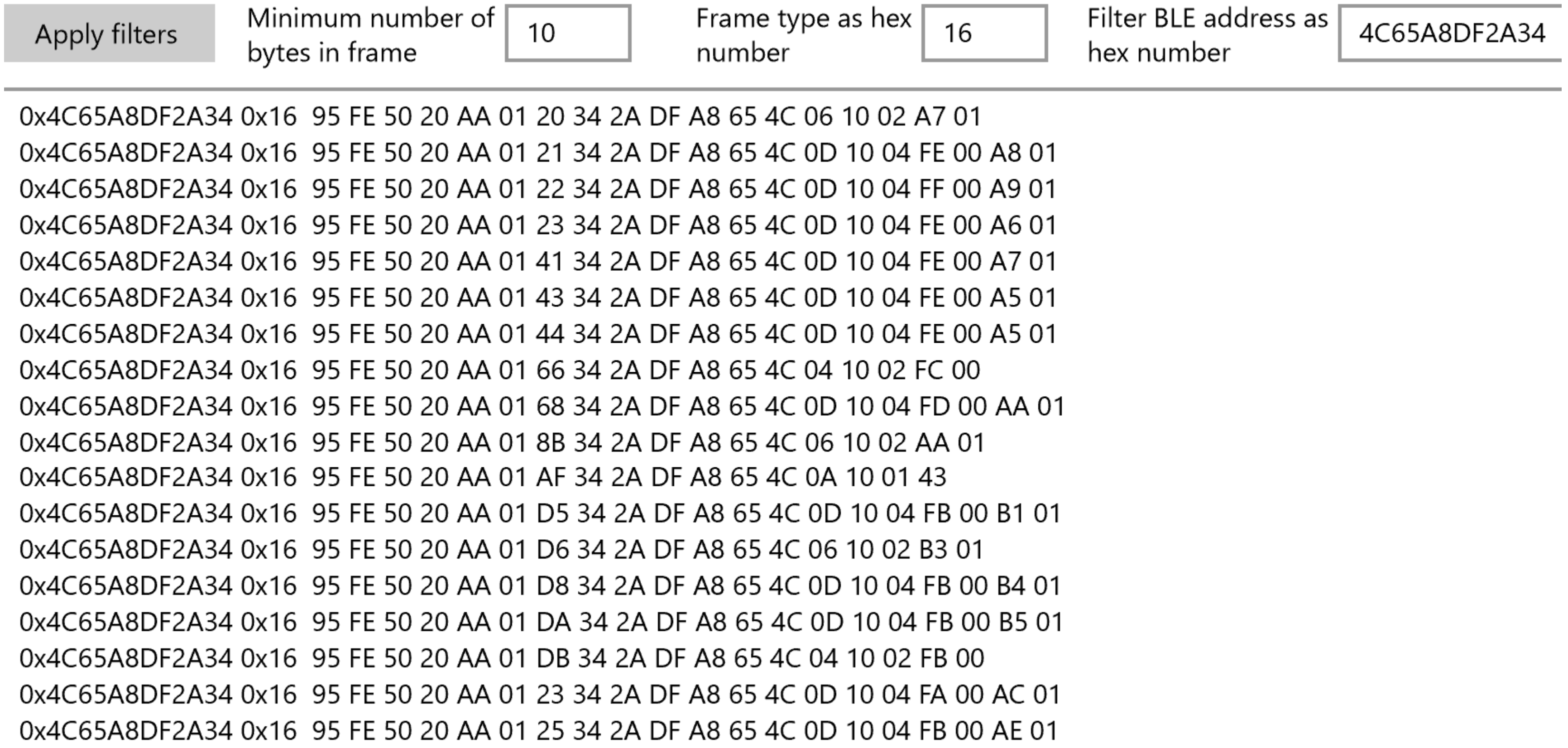
- Find the Bluetooth address
- Analyse the received frames
- Omit frames with constant bytes
- Look for frames with a constant startup sequence and differing bytes after
- Check this changing bytes for encoded temperature, humidity and battery level

Recording all frames of the Xiaomi device


Apply filters
Minimum number of bytes in frame
0
Frame type as hex number
Filter BLE address as hex number
4C65A

```
0x4C65A8DF2A34 0x16  FF FF D2 CF 08 EE E6 50
0x4C65A8DF2A34 0x1  06
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 E9 34 2A DF A8 65 4C 0D 10 04 FC 00 B5 01
0x4C65A8DF2A34 0x1  06
0x4C65A8DF2A34 0x16  FF FF D2 CF 08 EE E6 50
0x4C65A8DF2A34 0x1  06
0x4C65A8DF2A34 0x16  FF FF D2 CF 08 EE E6 50
0x4C65A8DF2A34 0x1  06
0x4C65A8DF2A34 0x16  FF FF D2 CF 08 EE E6 50
0x4C65A8DF2A34 0x1  06
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 7F 34 2A DF A8 65 4C 0D 10 04 FC 00 B0 01
0x4C65A8DF2A34 0x16  FF FF D2 CF 08 EE E6 50
0x4C65A8DF2A34 0x1  06
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 7C 34 2A DF A8 65 4C 04 10 02 FB 00
0x4C65A8DF2A34 0x1  06
0x4C65A8DF2A34 0x16  FF FF D2 CF 08 EE E6 50
```

Xiaomi device frames filtered



Apply filters | Minimum number of bytes in frame: 10 | Frame type as hex number: 16 | Filter BLE address as hex number: 4C65A8DF2A34

```
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 20 34 2A DF A8 65 4C 06 10 02 A7 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 21 34 2A DF A8 65 4C 0D 10 04 FE 00 A8 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 22 34 2A DF A8 65 4C 0D 10 04 FF 00 A9 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 23 34 2A DF A8 65 4C 0D 10 04 FE 00 A6 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 41 34 2A DF A8 65 4C 0D 10 04 FE 00 A7 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 43 34 2A DF A8 65 4C 0D 10 04 FE 00 A5 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 44 34 2A DF A8 65 4C 0D 10 04 FE 00 A5 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 66 34 2A DF A8 65 4C 04 10 02 FC 00
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 68 34 2A DF A8 65 4C 0D 10 04 FD 00 AA 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 8B 34 2A DF A8 65 4C 06 10 02 AA 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 AF 34 2A DF A8 65 4C 0A 10 01 43
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 D5 34 2A DF A8 65 4C 0D 10 04 FB 00 B1 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 D6 34 2A DF A8 65 4C 06 10 02 B3 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 D8 34 2A DF A8 65 4C 0D 10 04 FB 00 B4 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 DA 34 2A DF A8 65 4C 0D 10 04 FB 00 B5 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 DB 34 2A DF A8 65 4C 04 10 02 FB 00
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 23 34 2A DF A8 65 4C 0D 10 04 FA 00 AC 01
0x4C65A8DF2A34 0x16  95 FE 50 20 AA 01 25 34 2A DF A8 65 4C 0D 10 04 FB 00 AE 01
```

Advertising beacon sensors pros and cons

- You cannot be sure that you receive data from the right sensor, you cannot detect forged packets on system level
- 1: n Sensor⇔ Consumer
- Data are always public to anybody, that can intercept the Bluetooth signal
- Hacks of unknown sensors are in most cases easy and give fun

Windows IoT Core on Raspberry Pi

Monitor with HDMI connector
USB Keyboard, USB Mouse
SD Card 16 GB
Windows 10 IoT Core Dashboard
Windows IoT Remote Client => Issue on Raspbi 3 Nano RDP Server not working

Install Windows IoT Core on Raspberry Pi

- Windows 10 IoT Core Dashboard
- Select OS Build
- Flash the SD Card
- Start Raspberry by plugin power supply
- Connect to network Wired or wireless
- Open Windows Device Portal
- Open <IP of Raspberry>:8080 in Browser

Developing, deploying, debugging and running of UWP applications on Raspberry Pi

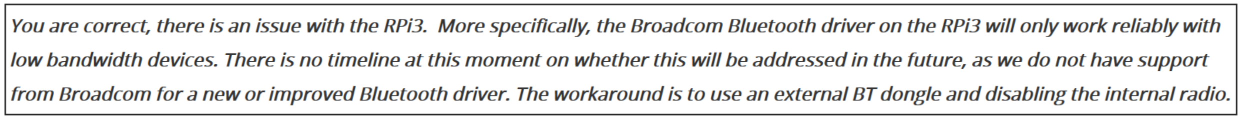
- UWP applications run 1:1 on Windows IoT core
- But keep the UI as simple as possible
- Select ARM as Target
- Update Debugging options:
- Set IP address of Raspberry, set authentication mode to Universal
- Start debugging (Code developed and tested on Win10)
- Press Run

Issues with Windows IoT core on Raspberry Pi 3

Bluetooth not properly working in general and in particular for GATT sensors

Blog entry with comment from Microsoft

http://embedded101.com/Blogs/David-Jones/entryid/790/Win-10-IoT-Core-Raspberry-Pi3-Bluetooth-Driver-Issue-Workaround



You are correct, there is an issue with the RPi3. More specifically, the Broadcom Bluetooth driver on the RPi3 will only work reliably with low bandwidth devices. There is no timeline at this moment on whether this will be addressed in the future, as we do not have support from Broadcom for a new or improved Bluetooth driver. The workaround is to use an external BT dongle and disabling the internal radio.

- The 4 $ solution

More issues with Windows IoT core on Raspberry Pi 3

Network connection lost after some time

- Disable power management of Wi-Fi adapter

Program stops from time to time in long term test at 2:00 am

- Disable Windows Update

All these actions improved stability, but it still was not perfect, from time to time Raspberry Pi stopped working, even if System Idle Process was never < 90 %. Pi showed an error on booting, flashing of the SD card was necessary

More issues with Windows IoT core on Raspberry Pi 3

After some debug sessions, I noticed that the Pi was quite hot, and if I waited for some time Windows IoT would boot successfully without flashing the SD card.

Was there a heating problem?

Now if you're on the right track,

you will find a cooling solution

designed by Microsoft

Building Microsoft's cooling rig for the Pi 3.
Image: Microsoft

More issues with Windows IoT core on Raspberry Pi 3

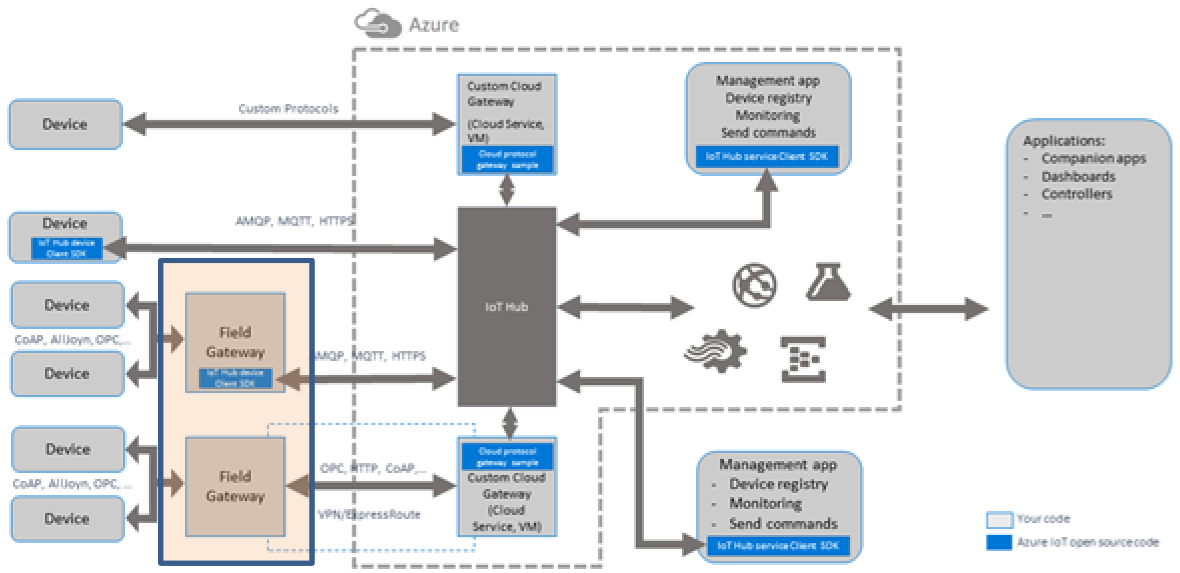
So: Use a metal case, add heatsinks, and eventually a fan

Or use this device to avoid all issues from the first

Dragonboard 410C Qualcomm Sapdragon 410E

- Avoiding all the issues of Raspbi and Windows IoT Core
- Not available in CH, must order in US
- Raspbi has more options USB, wired network

Where are we with our “sensor network”?



Azure IoT Central

- New SAAS solution for IoT
- Create IoT applications without coding
- Provides secure connections from device to cloud and vice versa

Azure IoT Central

The agenda

- Transmit our sensor values to IoT Central
- Display sensor values in IoT Central as charts
- Store our sensor values in Azure Blob Storage
- Create rules for monitor sensor values, create alarm, if given thresholds are violated
- Sends messages from Raspberry PI to IoT Central
- Switch on and off device and sensors on Pi by commands from IoT Central
- Simulate configuration update for our sensor network on Pi by commands from IoT Central

Azure IoT Central

Prerequisites

Create an Azure account

Create Azure IoT Central Application

Install the tools for generating an
IoT Central connection key:

- Install the DPS key generator by using the following command:
  ```
  npm i -g dps-keygen
  ```
- Download the dps_cstr tool from github
  https://github.com/Azure/dps-keygen/blob/master/bin/windows/dps_cstr.zip
  ```
  dps_cstr <scope_id> <device_id> <Primary Key>
  ```
- Add Nuget Package Microsoft.Azure.Devices.Client to your client project